**代理模式**

参考：<https://blog.csdn.net/yuanzeyao/article/details/7442286>

<https://www.cnblogs.com/cenyu/p/6289209.html>

<https://blog.csdn.net/goskalrie/article/details/52458773>

1. 代理模式：给某一个对象提供一个代理，代理对象持有目标对象的引用，通过代理控制目标对象的访问，从而在实现目标对象功能的基础上，扩展目标对象的功能。
2. 抽象角色（ISubject）：声明了代理主题和真实主题的公共接口,使任何需要真实主题的地方都能用代理主题代替
3. 真实角色(RealSubject)：定义了代理角色所代表的具体对象
4. 代理角色(Proxy)：含有对真实主题的引用，从而可以在任何时候操作真实主题，代理主题和真实主题实现相同的接口，使它可以随时代替真实主题。代理主题在执行真是主题的操作时，附加其他操作，相当于实现了对真实主题的封装。
5. 静态代理
6. 抽象角色

public interface Manager {  
 void doSomething();  
}

1. 真实角色

public class Admin implements Manager {  
 public void doSomething() {  
 System.*out*.println("Admin do something.");  
 }  
}

1. 代理角色
2. 聚合方式实现代理角色

//以聚合方式实现的代理主题  
public class AdminPoly implements Manager{  
 private Admin admin;  
  
 public AdminPoly(Admin admin) {  
 super();  
 this.admin = admin;  
 }  
  
 public void doSomething() {  
 System.*out*.println("Log:admin操作开始");  
 admin.doSomething();  
 System.*out*.println("Log:admin操作结束");  
 }  
}

1. 继承方式实现代理角色

//继承式静态代理  
public class AdminProxy extends Admin {  
 @Override  
 public void doSomething() {  
 System.*out*.println("Log:admin操作开始");  
 super.doSomething();  
 System.*out*.println("Log:admin操作开始");  
 }  
}

1. 测试代码
2. 聚合方式实现代理模式——测试代码

Admin admin = new Admin();  
Manager manager = new AdminPoly(admin);  
manager.doSomething();

1. 继承方式实现代理角色——测试代码

AdminProxy proxy = new AdminProxy();  
proxy.doSomething();

1. 动态代理
2. JDK动态代理
3. Cglib动态代理